Open Scene Graph Keyboard and Stereo

In this exercise you will learn how to add some simple keyboard controls using a separate class. I suggest that if you want to have more control over input, including joystick controls, you look at DirectInput (this is much more complex, so not for the faint-hearted!). You will also look at basic stereo settings.

- Create a new Win32 console project called *OSG keyboard and stereo*
- Set the properties as for the lighting tutorial (including the libraries)
- Make sure you have downloaded the resources and put them in a "models" folder inside your project

Create geometry and display in the viewer

```
#include "stdafx.h"
#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osg/Node>
#include <osg/PositionAttitudeTransform>
#include <osg/ref_ptr>


nt _tmain(int argc, _TCHAR* argv[])
{
	// Declare a node which will serve as the root node
	// the rest of the scene "hangs" on this node

	osg::ref_ptr<osg::Group>  myRoot  = new osg::Group();

	//*************************  load an object in the scene
*********************//

	osg::ref_ptr<osg::Node> blueHex = osgDB::readNodeFile("models/lBlueHex.3ds");
	osg::ref_ptr<osg::PositionAttitudeTransform>  blueXform = new
osg::PositionAttitudeTransform();
	osg::Vec3 bluePos = osg::Vec3(0,1,0); // use a vector to set the position of
the object
	blueXform->setPosition(bluePos);
	blueXform->addChild(blueHex.get());
	myRoot->addChild(blueXform.get()); // adds it all to the root node


	//************************************************************//


		//*******set up the viewer to display your scene***************//

	osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene

	viewer.setSceneData( myRoot.get()); //assign the scene graph we created above
to this viewer

	viewer.setLightingMode(osg::View::LightingMode::HEADLIGHT); // give the scene
some lighting
```

```
	viewer.getCamera()->setClearColor(osg::Vec4(1.0,1.0,1.0,1.0)); // sets
background colour of scene

	viewer.getCamera()-
>setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,0),osg::Vec3(0.0,1.0,0), osg::Vec3(0.0, 0.0,
1.0));


	//*************************************************************//


	viewer.realize(); // start the viewer


	//************ add a title to the window ***********************//

	typedef osgViewer::Viewer::Windows Windows;
	Windows windows;

	viewer.getWindows(windows);
	windows[0]->setWindowName("UoP osg keyboard and stereo example");

	//*****************************************//



while (!viewer.done()) // until the end of the program
	{

		viewer.frame(); // update to next frame

	}


	return 0;
}
```

Build and test the project. You should have a window with a blue hex box in it.

Adding keyboard control

Now you need to add the new file to your project. Take a look at the code in this file to see where the keyboard is being read and passed back to your main program.

- Make sure the *keyInput2.h* file is in your project folder with your main .cpp file. Then add the following header

```
#include <osg/PositionAttitudeTransform>
#include <osg/ref_ptr>

#include "keyInput2.h"


int _tmain(int argc, _TCHAR* argv[])
{
```

- Now let's create an event handler to deal with the keyboard input. Add the following code into your main program. Remember to take care with line breaks – don't follow the tutorial document forced line breaks!

```
viewer.getWindows(windows);
    windows[0]->setWindowName("UoP osg keyboard and stereo example");

    //*****************************************//

    //********************** set up keyboard control  **************//


    // declare the event handler for keyboard input - we are using a simple custom
keyboard class which you will need to add to your project folder.
    keyInputStateType* keyPressState = new keyInputStateType;
    myKeyboardEventHandler* keyEventHandler = new
myKeyboardEventHandler(keyPressState);

    viewer.addEventHandler(keyEventHandler);


    //*********************************************************//


    while (!viewer.done()) // until the end of the program
    {
```

Build and test the project again. You should have a window with a blue hex box in it, exactly the same as before. Although we are now listening for keyboard input, we aren't doing anything with it. The next thing to do is create a function to do something with the key strokes.

- Add the following function declaration to the **top** of your file

```
#include "keyInput2.h"

// function definitions
osg::Vec3 keyListen(int, osg::ref_ptr<osg::PositionAttitudeTransform>);  //
receives keyboard input and acts on it


int _tmain(int argc, _TCHAR* argv[])
{


    return 0;
}
```

- And add the definition after your main program

```
 // function to check keyboard input and act accordingly.  Object is passed
through as ref, current position is updated and passed back to main program
```

```
osg::Vec3 keyListen(int key, osg::ref_ptr<osg::PositionAttitudeTransform>
moveObject)
{
    osg::Vec3 tempPos = moveObject->getPosition(); // get the current object
position and place it in a temporary vector

    switch(key)
    {
    case 37: // left arrow key
           tempPos[0] -= 0.01; // subtract from the x axis
           break;
    case 39:  // right arrow key
           tempPos[0] += 0.01; // add to the x axis
           break;
    case 38: // up arrow key
           tempPos[2] += 0.01; // add to the z axis
           break;
    case 40: // down arrow key
           tempPos[2] -= 0.01; // subtract from the z axis
           break;
    case 'f': //forwards
           tempPos[1] += 0.01; // add to y axis
           break;
    case 'b': //backwards
           tempPos[1] -= 0.01; // subtract from y axis
           break;
           break;
    }
    return (tempPos);
}
```

Build and test the project again. **You still won't see any response to key presses.** This is because we haven't yet called the function. When we do, we are going to use it to change the position of the blue box in our scene. We need to add a call to the function every frame

```
       while (!viewer.done()) // until the end of the program
       {
              if (keyPressState->keyInput)// if there has been a key press event -
this gets checked every frame
              {
                           bluePos = keyListen(keyPressState->key, blueXform); // the
function is going to set bluePos to a new value
                           blueXform->setPosition(bluePos); // update the object
position
                           keyPressState->keyInput = false; // reset the state to
wait for next keypress
                           keyPressState->key = 'z'; // reset the state for next
input
              }
              viewer.frame(); // update to next frame

       }
```

Now when you build and test the project, you can move the box around using the arrow keys and the 'f' and 'b' keys. Take a look at the code and

see if you can customise it to respond to different keys. Be creative, explore scaling, rotating etc (you may need to create some new variables, or even a new function.....).

Setting up stereo

Next you are going to put in some basic anaglyph (red/green) stereo. You should spend some time this week reading about stereo settings and how they work, and experimenting with more precise control of your camera view and stereo setup.

- Add the following code into your program

```
viewer.getCamera()->setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,0),osg::Vec3(0.0,1.0,0),
osg::Vec3(0.0, 0.0, 1.0));


	//*************************************************************//

	//************************ set up custom display for stereo *****************
//


	osg::DisplaySettings *myDisplay = new osg::DisplaySettings; // holds parameters
for the visual display
	myDisplay->setDefaults(); // initialise with default settings
// you may want to adjust the height and width to suit your monitor
	myDisplay->setScreenHeight(0.24);
	myDisplay->setScreenWidth(0.42);
	myDisplay->setScreenDistance(0.5);
	myDisplay->setEyeSeparation(0.06); // set for stereo viewing - you will need to
adjust this to suit your scene


	myDisplay->setStereoMode(osg::DisplaySettings::ANAGLYPHIC);
	myDisplay->setStereo(true);

	viewer.setDisplaySettings(myDisplay);

	//*************************************************************//


	viewer.setUpViewInWindow(50,50, 800, 800); // the first two parameters are the
x and y position, the last two are the height and width

	viewer.realize(); // start the viewer
```

You will find that as you use your keyboard controls to move the box backwards and forwards, you will reach a point at which the stereo settings seem about right. Experiment with eye separation to change this. Also, during this week, see if you can work out how to calculate the correct settings for your scene and display size. You will need to do some background reading!

You might find the following code snippets help to point you in the right direction, but you will have to read up on them.

```
viewer.getCamera()->getProjectionMatrixAsFrustum(fleft, fright, fbottom, ftop, fnear, ffar);

viewer.getCamera()->setProjectionMatrixAsFrustum(fleft, fright,fbottom, ftop, fnear, ffar);

myDisplay->setScreenHeight(SCREENHEIGHT);

myDisplay->setScreenWidth(SCREENWIDTH);

myDisplay->setScreenDistance(SCREENDISTANCE);
```